#This script contains functions for benchmarking. In this script, these functions are applied to transmission cost benchmarking, but could be applied to any single equation model.

#This script does the following:

#1. Accepts a raw dataset, mean scales and logs a user-selected set of variables

#2. Performs an fgls regression(s) of the user's choosing

#3. Benchmarks all the companies in the sample for all specifications during the most recent three years for each company

#4. Benchmarks any individual company (or companies) of the user's choosing for all specifications during all years

#5. Gets goodness of fit statistic for all models

#6. Calculates elasticities and marginal costs for a set of user-specified outputs

#7. Organizes results and saves them to a text file

#Load necessary libraries

```
library(panelAR)

#library(Ryacas)

library(msm)
```

#This function meanscales selected variables.

#Forecast values for certain companies may be included in the input dataset but excluded

#from the calculation of the means by using the incfc and fcvar options.

#The function returns a list containing two dataframes,

#the mean-scaled data and the list of means used to scale the data.

#The verbose option is provided for debugging purposes,

#changing it from its default to T will cause the function to print the class of the each variable being mean scaled,

#its name, and its mean, as the function loops over the variables specified by the user.

```
meanscale <- function(fulldata,
                      varlist,
```

```
                verbose = F,
                incfc = F,
                fcvar = NULL) {
  #Create data frame in which to store the means of the variables.
  meanframe <-
    data.frame(
      varname = varlist,
      varmean = rep(0, length(varlist)),
      stringsAsFactors = F
    )

  #Initialize vector of means
  tempmeans <- NULL

  #Create temporary datasets. This distinction is only meaningful if the incfc and fcvar options are
specified.
  newdata <- fulldata
  meandata <- fulldata

  #Check if forecast values have been included in the input data frame, and if so, drop them from the
dataset used to calculate the sample means.

  if (incfc == T) {
    fcvarmark <- which(names(fulldata) == fcvar)
    tempdrop <- which(fulldata[, fcvarmark] == 1)
    meandata <- meandata[-tempdrop, ]
    rm(tempdrop)
  }
```

```
  #Loop over variables to be mean-scaled
  for (i in varlist) {
    #Get location of current variable in the data frame
    varmark <- which(names(meandata) == i)
    if (verbose == T) {
      print(class(meandata[, varmark]))
    }
    #Calculate the mean of the current variable, excluding missing values from this calculation.
    tempmean <- mean(meandata[, varmark], na.rm = T)
    if (verbose == T) {
      print(c(i, tempmean))
    }
    #Append the mean of the current variable to the vector of means
    tempmeans <- c(tempmeans, tempmean)

    #Scale all observations data for the current variable (not necessarily just the data used to calculate the mean) by the sample mean.
    newdata[, varmark] <- fulldata[, varmark] / tempmean
    #Remove temporary variables from memory before next cycle of loop.
    rm(tempmean, varmark)
  }

  #Fill the column of the data frame of variable means with the vector constructed during the above loop.
  meanframe$varmean <- tempmeans

  #Remove temporary objects from memory.
  rm(tempmeans, meandata)
```

```
  #Create list of outputs.
  outputmean <- list(newdata, meanframe)

  #Return outputs.
  return(outputmean)

}

#This function logs selected variables
#The verbose option is provided for debugging purposes.
#Changing it from its default to T will cause the function to print the variable name and class
#as it loops over the variables specified by the user.
logvars <- function(fulldata, varlist, verbose = F) {
  #Create local data frame for new data.
  newdata <- fulldata
  varinfo <-
    data.frame(
      varname = varlist,
      check0 = rep(0, length(varlist)),
      stringsAsFactors = F
    )


  #Loop over variables to be logged
  for (i in 1:length(varlist)) {
    tempvar <- varlist[i]
    #Get location of current variable in input data frame
    varmark <- which(names(fulldata) == tempvar)
```

```
  #check if there are any zero values in the current variable
  check0 <- ifelse(length(which(fulldata[, varmark] == 0)) > 0, 1, 0)


  #print information on current variable if verbose was specified as true
  if (verbose == T) {
    print(c(tempvar, class(fulldata[, varmark]), check0))
  }
  #Replace current variable in the local data frame with the natural logarithm of that variable in the input data frame. If that variable has any zero values, add 1 to all values before logging.
  if (check0 == 0) {
    newdata[, varmark] <- log(fulldata[, varmark])
  }
  else{
    newdata[, varmark] <- log(1 + fulldata[, varmark])
  }
  #Remove temporary variables before next cycle of loop.
  varinfo$varname[i] <- tempvar
  varinfo$check0[i] <- check0
  rm(varmark, check0, tempvar)
 }
 #Return output data frame.
 outputlogs <- list(newdata, varinfo)
 return(outputlogs)
}



#This function performs fgls regressions of any number of dependent variables on any number of right hand side specifications.

#This function is primarily a wrapper for the panelAR function.
```

```
#The arguments are as follows:
# 1) alldata is the complete set of data for the regression
# 2) idvar is the name of the variable that uniquely identifies companies in the sample
# 3) tvar is the name of the variable that uniquely identifies time periods in the sample
# 4) actype is the form of the autocorrelation to be used during the fgls regressions
# 5) rhometh is the method to be used during the calculation of autocorrelation coefficients
# 6) hetmeth is the form of the panel heteroskedasticity
# 7) rspeclist is the list of right hand side specifications for the regressions
# 8) depvarlist is the vector of dependent variables for the regressions

#The methods supported by panelAR for actype are "none", "ar1" (common autocorrelation coefficient),
#and "psar1" (panel specific autocorrelation coefficients).

#The methods supported by panelAR for rhometh are
#"breg" (regression of residuals on lagged values), "scorr" (sample autocorrelation coefficient),
#"dw" (durbin-watson autocorrelation coefficient),
#"freg" (regression of residuals on leading values),
#"theil" (scaled sample autocorrelation coefficient), and "theil-nagar" (modified durbin watson
autocorrelation coefficient).

#The methods supported by panelAR for hetmeth are "none", "phet" (Huber-White sandwich standard
errors),
#"pcse" (panel-corrected standard errors), "pwls" (panel weighted least squares), "parks" (parks-
kmenta).
#This function returns a list of models ordered first by dependent variable and then by right hand side
specification.

allspecregpanelAR <-
  function(alldata,
           idvar,
```

```
        tvar,
        actype = "none",
        rhometh = "breg",
        hetmeth = "none",
        rspeclist,
        depvarlist) {
  #Initialize the list of models.
  outputmodels <- list(NULL)

  #Loop over dependent variables.
  for (i in 1:length(depvarlist)) {
    #Get current dependent variable.
    tempdep <- depvarlist[i]

    #Loop over right hand side specifications.
    for (j in 1:length(rspeclist)) {
      #Store current specification in a temporary variable.
      temprspec <- rspeclist[[j]]

      #Create temporary object containing the formula for the current dependent variable and current
right hand side specification.
      tempformula <- as.formula(paste(tempdep, "~", temprspec, sep = ""))
      #Print the formula currently in use by the function
      print(tempformula)
      if (actype == "none") {
        tempregdata <- alldata
      }
      if (actype != "none") {
        idvarmark <- which(names(alldata) == idvar)
```

```
  tempcomptab <- table(alldata[, idvarmark])
  tempexclocs <-
    which(alldata[, idvarmark] %in% as.numeric(names(which(
      tempcomptab == 1
    ))))
  if (length(tempexclocs) > 0) {
    tempregdata <- alldata[-tempexclocs, ]
  }
  if (length(tempexclocs) == 0) {
    tempregdata <- alldata
  }
  rm(idvarmark, tempcomptab, tempexclocs)
}
tempmodel <-
  panelAR(
    tempformula,
    data = tempregdata,
    panelVar = idvar,
    timeVar = tvar,
    autoCorr = actype,
    rhotype = rhometh,
    panelCorrMethod = hetmeth
  )

tempmodel$call$panelCorrMethod <- hetmeth
tempmodel$call$autoCorr <- actype
tempmodel$call$rhotype <- rhometh

#Fill first entry of the list of models for output if this is the first cycle of the routine.
```

```
      if (tempdep == depvarlist[1] & temprspec == rspeclist[1]) {
        outputmodels[[1]] <- tempmodel
      }
      #If this is not the first cycle of the routine, append the model just regressed to the list of models for output.
      if (tempdep != depvarlist[1] | temprspec != rspeclist[1]) {
        outputmodels[[length(outputmodels) + 1]] <- tempmodel
      }

      #Name the model appropriately (dependent variable name - specification number)
      names(outputmodels)[length(outputmodels)] <-
        paste(tempdep, " - Specification ", j, sep = "")

      #Remove temporary objects from memory before next cycle of loop.
      rm(tempformula, tempmodel, temprspec, tempregdata)

    }
    #Remove temporary objects from memory before next cycle of loop.
    rm(tempdep)

  }

  #Return list of models for output.
  return(outputmodels)

}
```

```
#This function performs benchmarking of all companies in an input sample called benchdata.

#The arguments for this function are as follows:

# 1) models is the list of all models to be used for benchmarking

# 2) alldata is the full dataset used during the regression of models passed to this function for
benchmarking

# 3) benchdata is the subset of alldata to be used for benchmarking purposes

# 4) unscale is a binary variable indicating whether actual and predicted costs should be

#    returned from their logged and mean scaled values to their actual dollar amounts

#    if unscale is true, varmeans is the data frame containing variable names and means returned by the
function meanscale

# 5) idvar is the name of the variable that uniquely identifies companies in the sample

# 6) tvar is the name of the variable that uniquely identifies time periods in the sample

# 7) namevar is an optional input containing the name of the column in benchdata where the names of
the companies to be benchmarked can be found

# 9) tstat is a binary variable specifying whether t statistics and p values should be calculated for the
predicted differences

allspecbench <-

  function(models,

           alldata,

           benchlength = 3,

           unscale = F,

           varmeans = NULL,

           idvar,

           tvar,

           namevar = NULL,

           tstat = F,

           rdigits = 3) {
```

```
#Initialize the list of data frames containing the benchmarking results
outbenchmarks <- list(NULL)

#Get the location of the company id variable in the benchmarking data frame
idvarmark <- which(names(alldata) == idvar)
#Get the location of the time period variable in the benchmarking data frame
tvarmark <- which(names(alldata) == tvar)
#Get a vector of the company ids for the companies to be benchmarked
tempcomplist <- unique(alldata[, idvarmark])


#Initialize a vector of the latest year in the sample for each company
benchmaxyear <- NULL
#Loop over companies
for (i in tempcomplist) {
  #Get the rows containing the data of the company currently under consideration
  comptemplocs <- which(alldata[, idvarmark] == i)
  #Get the value of the latest year available for the company currently under consideration
  comptempmax <- max(alldata[comptemplocs, tvarmark])
  #Add the current company's latest year to the vector of all companies' latest years
  benchmaxyear <- c(benchmaxyear, comptempmax)
  rm(comptemplocs, comptempmax)
}

#Create a data frame containing the list of companies and their latest available years and duplicate it
allbenchlistraw <- as.data.frame(cbind(tempcomplist, benchmaxyear))
allbenchlist <- allbenchlistraw
if (benchlength > 1) {
  for (i in 2:benchlength) {
```

```
      tempbenchlist <- allbenchlistraw
      #In each duplicate, subtract the appropriate number of years from the column containing the value of the latest available year for each company
      tempbenchlist$benchmaxyear <- tempbenchlist$benchmaxyear - (i - 1)
      #Append the duplicates to the original data frame
      allbenchlist <- rbind(allbenchlist, tempbenchlist)
      rm(tempbenchlist)
    }
  }
  #Create unique ids for each company-year observation in the data frame just created
  benchkeepids <-
    paste(allbenchlist$tempcomplist, allbenchlist$benchmaxyear, sep = "_")

  #Create unique ids for each company-year observation in the full sample data frame
  allrealid <- paste(alldata[, idvarmark], alldata[, tvarmark], sep = "_")

  #Create a new data frame containing the data of the companies during only the benchmarking period
  benchdata <- alldata[which(allrealid %in% benchkeepids), ]

  rm(benchmaxyear,
     allbenchlistraw,
     allbenchlist,
     benchkeepids,
     allrealid)
```

```
  #Loop over the list of models
  for (i in 1:length(models)) {
    #Print the name of the model currently in use
    print(names(models)[i])
    #Get the dependent variable name for the model currently in use
    tempdepvarname <- unlist(strsplit(names(models)[i], " - "))[1]
    #Get the location of the dependent variable currently in use in the benchmarking data frame
    tempdepmark <- which(names(benchdata) == tempdepvarname)
    #Store the current model in a temporary variable
    tempmodel <- models[[i]]
    #Get the names of the right hand side variables currently in use for the model
    tempindvarnamesm <- names(tempmodel$coefficients[-1])
    #Initialize the formula for use in the benchmarking for the current model
    tempformula <- paste(tempdepvarname, "~", tempindvarnamesm[1], sep = "")

    #If there is more than one right hand side variable, complete the formula for benchmarking the
current model
    if (length(tempindvarnamesm) > 1) {
      for (j in 2:length(tempindvarnamesm)) {
        tempformula <- paste(tempformula, tempindvarnamesm[j], sep = "+")
      }
    }

    #Convert the formula currently stored as a string to a formula
    tempformula <- as.formula(tempformula)

    #Initialize vectors for the observed means, predicted means, and difference means during the
benchmarking period
    tempcompobsmeans <- NULL
```

```
    tempcomppredmeans <- NULL
    tempcompdiffmeans <- NULL

    #Initialize vectors for t statistics and p values corresponding to the differences between observed and predicted values during the benchmarking period
    tempcomptstats <- NULL
    tempcomppvals <- NULL

    #Initialize vector for recording the number of years benchmarked
    tempcompyrsbench <- NULL

    #Get the form of the autocorrelation used in the model currently in use for benchmarking
    tempactype <- summary(tempmodel)$call$autoCorr
    #Get the method used to calculate the autocorrelation coefficients
    temprhometh <- summary(tempmodel)$call$rhotype
    #Get the form of the heteroskedasticity estimated by the model currently in use for benchmarking
    temphetmeth <- summary(tempmodel)$call$panelCorrMethod


    #Loop over  the companies to be benchmarked
    for (j in 1:length(tempcomplist)) {
     #Store the independent variable names for the current model in a temporary vector that may be modified later
      tempindvarnames <- tempindvarnamesm
      #Get all data except that of the company currently being benchmarked
      tempooslocs <- which(alldata[, idvarmark] != tempcomplist[j])
      tempoosdata <- alldata[tempooslocs, ]

      if (tempactype != "none") {
```

```
  tempcomptab <- table(tempoosdata[, idvarmark])
  tempexclocs <-
    which(tempoosdata[, idvarmark] %in% as.numeric(names(which(
      tempcomptab == 1
    ))))
  if (length(tempexclocs) > 0) {
    tempoosdata <- tempoosdata[-tempexclocs, ]
  }
  rm(tempcomptab, tempexclocs)
}



#Get the benchmarking period data for the company currently being benchmarked
tempbenchlocs <- which(benchdata[, idvarmark] == tempcomplist[j])
tempbenchdata <- benchdata[tempbenchlocs, ]

#Perform the desired fgls regression.
tempoosmodel <-
  panelAR(
    tempformula,
    data = tempoosdata,
    panelVar = idvar,
    timeVar = tvar,
    autoCorr = tempactype,
    rhotype = temprhometh,
    panelCorrMethod = temphetmeth
  )
```

```
#Correct components of the model's call
tempoosmodel$call$panelCorrMethod <- temphetmeth
tempoosmodel$call$autoCorr <- tempactype
tempoosmodel$call$rhotype <- temprhometh

#Get the predicted values from the current, reestimated model during the benchmarking period for the company being benchmarked
temppred <- predict(tempoosmodel, newdata = tempbenchdata, se.fit = F)$fit

#Create a temporary data frame containing the values of company id, time period, observed dependent variable, and predicted dependent variable for the company currently being benchmarked
tempdata <-
  cbind(tempbenchdata[, c(idvarmark, tvarmark, tempdepmark)], temppred)
tempdata <- as.data.frame(tempdata)

#Get the mean of the unlogged observed and predicted values during the benchmarking period, as well as the mean percent difference (calculated logarithmically)
tempobsmean <- mean(exp(tempdata[, 3]))
temppredmean <- mean(exp(tempdata[, 4]))
tempdiffmean <- mean(tempdata[, 3] - tempdata[, 4])

#Get the number of years benchmarked
tempyrsbench <- min(c(dim(tempbenchdata)[1], benchlength))
```

```
    #Check if tstat is true, and if so, calculate t statistics and p values for the mean difference between the observed and predicted values
    if (tstat == T) {
     #This section of the code is included for handling interactive and squared terms included in the model but not found in the data frame
    #This section identifies such terms, renames them systematically, and creates columns for them in the tempoosdata and tempbenchdata data frames
    #These steps are necessary to ensure proper handling of matrix operations later in the code for calculating t statistics

    #Loop over the names of all independent variables
    for (k in 1:length(tempindvarnames)) {
     #Check whether the independent variable under consideration is an squared term
     tempIcheck <- ifelse(substr(tempindvarnames[k], 1, 2) == "I(", 1, 0)

     #If the independent variable under consideration is a squared term, standardize its naming
     if (tempIcheck == 1) {
      #Cut the "I(" and ")" off either end of the squared term's string components
      tempIvar <- substr(tempindvarnames[k], 3, nchar(tempindvarnames[k]))
      tempIvar <- substr(tempIvar, 1, nchar(tempIvar) - 1)
     #Replace the old name of the squared term with a new variable name whose form matches the names of interactive terms
      tempindvarnames[k] <- tempIvar
      #Remove the components of the squared term from memory
      rm(tempIvar)
     }
```

```
tempsplit <- unlist(strsplit(tempindvarnames[k], ":"))
tempccheck <- ifelse(length(tempsplit) > 1, 1, 0)
if (tempccheck == 1) {
 tempindvarnames[k] <- gsub(":", "*", tempindvarnames[k])
}

#Check whether the independent variable under consideration is a squared or interaction term
tempcheck <- max(tempIcheck, tempccheck)

#If the independent variable under consideration is a squared or interaction term, create a new variable in both tempoosdata and tempbenchdata with the appropriate name and values
if (tempcheck == 1) {
 #Create variables in both data frames with the values of specified squared or interaction terms and name them appropriately
 tempbenchdata$tempint <-
  with(tempbenchdata, eval(parse(text = tempindvarnames[k])))
 names(tempbenchdata)[dim(tempbenchdata)[2]] <- tempindvarnames[k]
 tempoosdata$tempint <-
  with(tempoosdata, eval(parse(text = tempindvarnames[k])))
 names(tempoosdata)[dim(tempoosdata)[2]] <- tempindvarnames[k]
}
#Remove temporary objects from memory before next cycle of loop
rm(tempcheck, tempIcheck, tempccheck)
}

#Get locations of independent variables in the benchmarking and out of sample data frames
indvarlocs <- match(tempindvarnames, names(tempbenchdata))
```

```
#Create matrices containing the out of sample dependent variable values, independent variable values, and the current model's coefficients

tempoosdepmat <- as.matrix(tempoosdata[, tempdepmark])

tempoosindmat <- as.matrix(cbind(1, tempoosdata[, indvarlocs]))

tempcoef <-

  matrix(tempoosmodel$coefficients,

       nrow = length(tempoosmodel$coefficients))

#Get the current model's residual degrees of freedom and variance covariance matrix

tempdf <- tempoosmodel$df.residual

tempvcm <- tempoosmodel$vcov

#Create a matrix containing the independent variable values for the company currently being benchmarked

temptdata <- as.matrix(cbind(1, tempbenchdata[, indvarlocs]))

#Create a matrix containing the in-sample residuals for the current model

tempres <- tempoosdepmat - tempoosindmat %*% tempcoef

#Get the mean squared error for the current model

tempe2 <- t(tempres) %*% tempres / tempdf


#Get the number of time periods currently being benchmarked

temptperiods <- dim(tempbenchdata)[1]


#Create a matrix of 1's divided by the number of time periods currently being benchmarked, for use in the calculations that follow

temponea <- matrix(rep(1 / temptperiods, temptperiods), nrow = 1)


#Calculate the variance of the forecast average

tempSEEOA <-

  tempe2 / temptperiods + temponea %*% (temptdata %*% tempvcm %*% t(temptdata)) %*%

  t(temponea)
```

```
#Calculate the standard error of the forecast average
tempsetcA <- sqrt(tempSEEOA)

#Calculate the t statistic of the current average difference
temptval <- tempdiffmean / tempsetcA

#Append the current t statistic to the current list of t statistics
tempcomptstats <- c(tempcomptstats, temptval)

#Calculate the p-value of the current t statistic
temppval <- 2 * pt(abs(temptval), tempdf, lower.tail = F)

#Append the current p-value to the current list of p-values
tempcomppvals <- c(tempcomppvals, temppval)

#Remove temporary objects from memory before next cycle of loop
rm(
  indvarlocs,
  tempdf,
  tempvcm,
  temptdata,
  tempres,
  tempe2,
  temptperiods,
  temponea,
  tempSEEOA,
  tempsetcA,
  temptval,
  temppval,
```

```
    tempcoef,
    tempoosdepmat,
    tempoosindmat
  )
 }
 #Append current observed mean to current list of observed means
 tempcompobsmeans <- c(tempcompobsmeans, tempobsmean)
 #Append current predicted mean to current list of predicted means
 tempcomppredmeans <- c(tempcomppredmeans, temppredmean)
 #Append current mean difference to current list of mean differences
 tempcompdiffmeans <- c(tempcompdiffmeans, tempdiffmean)
 #Append current years benchmarked to current list of years benchmarked
 tempcompyrsbench <- c(tempcompyrsbench, tempyrsbench)
 #Remove temporary objects from memory before next cycle of loop
 rm(
  tempdata,
  tempobsmean,
  temppredmean,
  tempoosdata,
  tempbenchdata,
  tempbenchlocs,
  tempooslocs,
  tempoosmodel,
  tempyrsbench,
  tempdiffmean,
  tempindvarnames
 )
}
```

```
    #If unscale was specified as true, unscale values of the observed and predicted means for the current model
    if (unscale == T) {
      #Get the row of the current dependent variable in the varmeans data frame
      tempmeanloc <- which(varmeans$varname == tempdepvarname)
      #Store the mean of the current dependent variable
      tempdepmean <- varmeans$varmean[tempmeanloc]
      #Un mean scale the observed and predicted means for all companies being benchmarked
      tempcompobsmeans <- tempcompobsmeans * tempdepmean
      tempcomppredmeans <- tempcomppredmeans * tempdepmean
      #Remove temporary objects from memory before next cycle of loop
      rm(tempmeanloc, tempdepmean)
    }

    #Create component of output list, depending on whether tstat was specified as true or false
    if (tstat == T) {
      #Create a data frame containing the benchmarked company ids, observed means, predicted means, mean differences, t statistics, p values, and years benchmarked
      tempbench <-
        cbind(
          tempcomplist,
          tempcompobsmeans,
          tempcomppredmeans,
          tempcompdiffmeans,
          tempcomptstats,
          tempcomppvals,
          tempcompyrsbench
        )
      tempbench <- as.data.frame(tempbench)
```

```
#Name the columns appropriately
names(tempbench) <-
  c(idvar,
    "observed",
    "predicted",
    "diff",
    "tstat",
    "pvalue",
    "no. years")
#Round the columns appropriately and fix 0 pvalue results
tempbench$observed <- round(tempbench$observed, 0)
tempbench$predicted <- round(tempbench$predicted, 0)
tempbench$diff <- round(tempbench$diff, rdigits)
tempbench$tstat <- round(tempbench$tstat, rdigits)
tempbench$pvalue <- round(tempbench$pvalue, rdigits)
p0locs <- which(tempbench$pvalue == 0)
tempbench$pvalue <- as.character(tempbench$pvalue)
pvalchars <- nchar(tempbench$pvalue)
pval0s <- rdigits + 2 - pvalchars
for (j in 1:length(pval0s)) {
  tempbench$pvalue[j] <-
    ifelse(
      nchar(tempbench$pvalue[j]) != rdigits + 2,
      paste(
        tempbench$pvalue[j],
        paste(rep(0, pval0s[j]), sep = "", collapse = ""),
        sep = ""
      ),
      tempbench$pvalue[j]
```

```
      )
    }
    if (length(p0locs) > 0) {
     r0val <- paste("<", 10 ^ -rdigits, sep = "")
     tempbench$pvalue[p0locs] <- r0val
     rm(r0val)
    }
    #Remove temporary objects from memory before next cycle of loop
    rm(
     tempcompobsmeans,
     tempcomppredmeans,
     tempcompdiffmeans,
     tempcomptstats,
     tempcomppvals,
     tempcompyrsbench,
     p0locs,
     pvalchars,
     pval0s
    )
   }

   if (tstat == F) {
    #Create a data frame containing the benchmarked company ids, observed means, predicted means,
mean differences, and years benchmarked
    tempbench <-
     cbind(
      tempcomplist,
      tempcompobsmeans,
      tempcomppredmeans,
```

```
      tempcompdiffmeans,
      tempcompyrsbench
    )
  tempbench <- as.data.frame(tempbench)
  #Name the columns appropriately
  names(tempbench) <- c(idvar, "observed", "predicted", "diff", "no. years")
  tempbench$observed <- round(tempbench$observed, 0)
  tempbench$predicted <- round(tempbench$predicted, 0)
  tempbench$diff <- round(tempbench$diff, rdigits)
  #Remove temporary objects from memory before next cycle of loop
  rm(tempcompobsmeans,
    tempcomppredmeans,
    tempcompdiffmeans,
    tempcompyrsbench)
}
```

```
#Check if namevar was specified, and if so, add a column to the benchmarking results data frame containing the names of the companies benchmarked
if (length(namevar) > 0) {
  #Match the company ids to their first location in the benchmarking data frame
  tempnamelocs <- match(tempbench[, 1], benchdata[, idvarmark])
  #Get the location of the variable in the benchmarking data frame containing company names
  namevarmark <- which(names(benchdata) == namevar)
  #Add a new column to the benchmarking results data frame containing the names of the companies benchmarked
  tempbench$company <- benchdata[tempnamelocs, namevarmark]
  #Rearrange the columns so that company is the second column is the benchmarking results data frame
```

```
  tempbenchnewcols <- c(1, dim(tempbench)[2], 2:(dim(tempbench)[2] - 1))
  tempbench <- tempbench[, tempbenchnewcols]
  #Remove temporary objects from memory before next cycle of loop
  rm(tempnamelocs, namevarmark, tempbenchnewcols)
 }

 #Sort the benchmarking results data frame by difference, then (if necessary) by company id
 tempbench <- tempbench[order(tempbench$diff, tempbench[, 1]), ]

 #Store the benchmarking results data frame for the current model in the list of result sets for output,
and name it appropriately
 outbenchmarks[[i]] <- tempbench
 names(outbenchmarks)[i] <-
  paste(names(models)[i],
      " - up to ",
      benchlength,
      " year average results",
      sep = "")
 #Remove temporary objects from memory before next cycle of loop
 rm(
  tempmodel,
  tempdepmark,
  tempdepvarname,
  temppred,
  tempactype,
  tempbench,
  temprhometh,
  temphetmeth,
  tempformula,
```

```
      tempindvarnamesm
    )

    #Reset row names of current benchmarking results to their default
    row.names(outbenchmarks[[i]]) <-
      as.character(1:length(row.names(outbenchmarks[[i]])))

  }
  #Remove temporary objects from memory
  rm(idvarmark, tvarmark, tempcomplist)
  #Return list of benchmarking results
  return(outbenchmarks)
}
```

```
#This function calculates a pseudo R-squared for fgls models with heteroskedasticity and
autocorrelation corrections
#models is the list of models for which the pseudo R-squared should be calculated
getgof <- function(models) {
```

```
  #Initalize list of pseudo R-squared values
  goflist <- list(NULL)
  #Loop over models
  for (i in 1:length(models)) {
    #Get the current model's fitted values and residuals, convert them to data frames, and standardize
their order (sometimes the order of these outputs is not identical to their order in the input data frame)
    tempfitval <- models[[i]]$fitted.values
    tempfitresid <- models[[i]]$residuals
    tempfitval <- as.data.frame(tempfitval)
    tempfitresid <- as.data.frame(tempfitresid)
    tempfitval <- tempfitval[order(as.numeric(row.names(tempfitval))), ]
    tempfitresid <-
      tempfitresid[order(as.numeric(row.names(tempfitresid))), ]

    #Get the values of the observed dependent variable and the predicted dependent variable
    tempyhat <- tempfitval
    tempytrue <- tempfitval + tempfitresid
    #Calculate the pseudo R-squared for the current model, store it as data frame, and name name it
appropriately
    temppseudoR2 <- cor(tempyhat, tempytrue) ^ 2
    temppseudoR2 <- as.data.frame(temppseudoR2)
    names(temppseudoR2) <- "Pseudo R-Squared"
    #Store the pseudo R-squared of the current model in the list of values for output
    goflist[[i]] <- temppseudoR2

    #Remove temporary objects from memory before next cycle of loop
    rm(tempfitval,
      tempfitresid,
      tempyhat,
```

```
    tempytrue,
    temppseudoR2)
  }
  #Return the list of pseudo R-squared values
  return(goflist)
}
```

#This function performs benchmarking of one company during all the years for which it has data in the full sample data set

#The arguments for this function are as follows:

```
# 1) models is the list of all models to be used for benchmarking

# 2) alldata is the full dataset used during the regression of models passed to this function for benchmarking

# 3) companyid is the value of the variable specified as idvar for the company to be benchmarked (i.e. the company's pegid, pseid, eiid, etc)

# 4) unscale is a binary variable indicating whether actual and predicted costs should be returned from their logged and mean scaled values to their actual dollar amounts

#    if unscale is true, varmeans is the data frame containing variable names and means returned by the function meanscale

# 5) idvar is the name of the variable that uniquely identifies companies in the sample

# 6) tvar is the name of the variable that uniquely identifies time periods in the sample

# 7) compname is an optional input containing the name company to be benchmarked

# 9) tstat is a binary variable specifying whether t statistics and p values should be calculated for the predicted differences

singcompbench <-

 function(models,

          alldata,

          companyid,

          idvar,

          tvar,

          unscale = F,

          varmeans = NULL,

          compname = NULL,

          tstat = F,

          rdigits = 3) {

   #Initialize list of benchmarking results for output

   outbenchmarks <- list(NULL)

   #Get location of the the company id variable in the full sample data frame

   idvarmark <- which(names(alldata) == idvar)

   #Get location of the the time period variable in the full sample data frame
```

```
  tvarmark <- which(names(alldata) == tvar)


  #Loop over models
  for (i in 1:length(models)) {
   #Initialize data frame of benchmarking results
   tempresults <- NULL
   #Get the name of the dependent variable in the model currently under consideration
   tempdepvarname <- unlist(strsplit(names(models)[i], " - "))[1]
   #Get the location of the current dependent variable in the full sample data frame
   tempdepmark <- which(names(alldata) == tempdepvarname)
   #Store the current model in a temporary object
   tempmodel <- models[[i]]
   #Get the names of the right hand side variables currently in use for the model
   tempindvarnamesm <- names(tempmodel$coefficients[-1])
   #Initialize the formula used by the current model
   tempformula <- paste(tempdepvarname, "~", tempindvarnamesm[1], sep = "")
   #If there is more than one right hand side variable, complete the formula for benchmarking the
current model
   if (length(tempindvarnamesm) > 1) {
    for (j in 2:length(tempindvarnamesm)) {
     tempformula <- paste(tempformula, tempindvarnamesm[j], sep = "+")
    }
   }
   tempformula <- as.formula(tempformula)


   #Get the form of the autocorrelation used in the model currently in use for benchmarking
   tempactype <- summary(tempmodel)$call$autoCorr
   #Get the method used to calculate the autocorrelation coefficients
   temprhometh <- summary(tempmodel)$call$rhotype
```

```
    #Get the form of the heteroskedasticity estimated by the model currently in use for benchmarking
    temphetmeth <- summary(tempmodel)$call$panelCorrMethod

    #Store the independent variable names for the current model in a temporary vector that may be
modified later
    tempindvarnames <- tempindvarnamesm



    #Get all data except that of the company to be benchmarked
    tempooslocs <- which(alldata[, idvarmark] != companyid)
    tempoosdata <- alldata[tempooslocs, ]
    #Get all data for the company to be benchmarked
    tempbenchlocs <- which(alldata[, idvarmark] == companyid)
    tempbenchdata <- alldata[tempbenchlocs, ]

    tempoosmodel <-
      panelAR(
        tempformula,
        data = tempoosdata,
        panelVar = idvar,
        timeVar = tvar,
        autoCorr = tempactype,
        rhotype = temprhometh,
        panelCorrMethod = temphetmeth
      )

    #Correct components of the model's call
    tempoosmodel$call$panelCorrMethod <- temphetmeth
    tempoosmodel$call$autoCorr <- tempactype
```

```
tempoosmodel$call$rhotype <- temprhometh



#Calculate the predicted values from the current, reestimated model for the company being benchmarked

temppred <- predict(tempoosmodel, newdata = tempbenchdata, se.fit = F)$fit



#Fill the temporary data frame for results with the values of company id, time period, observed dependent variable, and predicted dependent variable, and ensure that it is a data frame

tempresults <-

 cbind(tempbenchdata[, c(idvarmark, tvarmark, tempdepmark)], temppred)

tempresults <- as.data.frame(tempresults)



#Store these results in a temporary object

tempdata <- tempresults



#Calculate the unlogged values of the observed dependent variable and the predicted dependent variable

tempobs <- exp(tempdata[, 3])

temppred <- exp(tempdata[, 4])



#Check if tstat is true, and if so, calculate t statistics and p values for the differences between the observed and predicted values

if (tstat == T) {

 #This section of the code is included for handling interactive and squared terms included in the model but not found in the data frame

 #This section identifies such terms, renames them systematically, and creates columns for them in the tempoosdata and tempbenchdata data frames

 #These steps are necessary to ensure proper handling of matrix operations later in the code for calculating t statistics
```

```
#Loop over the names of all independent variables
for (k in 1:length(tempindvarnames)) {
 #Check whether the independent variable under consideration is an squared term
 tempIcheck <- ifelse(substr(tempindvarnames[k], 1, 2) == "I(", 1, 0)

 #If the independent variable under consideration is a squared term, standardize its naming
 if (tempIcheck == 1) {
  #Cut the "I(" and ")" off either end of the squared term's string components
  tempIvar <- substr(tempindvarnames[k], 3, nchar(tempindvarnames[k]))
  tempIvar <- substr(tempIvar, 1, nchar(tempIvar) - 1)
  #Replace the old name of the squared term with a new variable name whose form matches the names of interactive terms
  tempindvarnames[k] <- tempIvar
  #Remove the components of the squared term from memory
  rm(tempIvar)
 }

 tempsplit <- unlist(strsplit(tempindvarnames[k], ":"))
 tempccheck <- ifelse(length(tempsplit) > 1, 1, 0)
 if (tempccheck == 1) {
  tempindvarnames[k] <- gsub(":", "*", tempindvarnames[k])
 }

 #Check whether the independent variable under consideration is a squared or interaction term
 tempcheck <- max(tempIcheck, tempccheck)

 #If the independent variable under consideration is a squared or interaction term, create a new variable in both tempoosdata and tempbenchdata with the appropriate name and values
```

```
        if (tempcheck == 1) {
          #Create variables in both data frames with the values of specified squared or interaction terms
and name them appropriately
          tempbenchdata$tempint <-
            with(tempbenchdata, eval(parse(text = tempindvarnames[k])))
          names(tempbenchdata)[dim(tempbenchdata)[2]] <- tempindvarnames[k]
          tempoosdata$tempint <-
            with(tempoosdata, eval(parse(text = tempindvarnames[k])))
          names(tempoosdata)[dim(tempoosdata)[2]] <- tempindvarnames[k]
        }
        #Remove temporary objects from memory before next cycle of loop
        rm(tempcheck, tempIcheck, tempccheck)
      }
      #Get locations of independent variables in the benchmarking and out of sample data frames
      indvarlocs <- match(tempindvarnames, names(tempbenchdata))


      #Create matrices containing the out of sample dependent variable values, independent variable
values, and the current model's coefficients
      tempoosdepmat <- as.matrix(tempoosdata[, tempdepmark])
      tempoosindmat <- as.matrix(cbind(1, tempoosdata[, indvarlocs]))
      tempcoef <-
        matrix(tempoosmodel$coefficients,
               nrow = length(tempoosmodel$coefficients))
      #Get the current model's residual degrees of freedom and variance covariance matrix
      tempdf <- tempoosmodel$df.residual
      tempvcm <- tempoosmodel$vcov
      #Create a matrix containing the independent variable values for the company currently being
benchmarked
      temptdata <- as.matrix(cbind(1, tempbenchdata[, indvarlocs]))
```

```
#Create a matrix containing the in-sample residuals for the current model
tempres <- tempoosdepmat - tempoosindmat %*% tempcoef
#Get the prediction component of the forecast variance
tempZ <- temptdata %*% tempvcm %*% t(temptdata)
#Get the mean squared error for the current model
tempe2 <- t(tempres) %*% tempres / tempdf
#Calculate the forecast variance for all years for the company being benchmarked
tempSEEOY <- tempe2 + diag(tempZ)
#Calculate the forecast standard error for all years for the company being benchmarked
tempsetcY <- sqrt(tempSEEOY)

#Calculate t statistics for the differences in all years for the company being benchmarked
temptvals <- (tempdata[, 3] - tempdata[, 4]) / tempsetcY
#Calculate p values for the t statistics in all years for the company being benchmarked
temppvals <- 2 * pt(abs(temptvals), tempdf, lower.tail = F)


#Remove temporary objects from memory before next cycle of loop
rm(
 indvarlocs,
 tempdf,
 temptdata,
 tempres,
 tempZ,
 tempe2,
 tempSEEOY,
 tempsetcY,
 tempindvarnames,
 tempoosdepmat,
```

```
    tempoosindmat,
    tempvcm,
    tempcoef
  )
 }

 #If unscale was specified as true, unscale values of the observed and predicted dependent variables for the current model
 if (unscale == T) {
  #Get the row of the current dependent variable in the varmeans data frame
  tempmeanloc <- which(varmeans$varname == tempdepvarname)
  #Store the mean of the current dependent variable
  tempdepmean <- varmeans$varmean[tempmeanloc]
  #Un mean scale the observed and predicted means for all companies being benchmarked
  tempobs <- tempobs * tempdepmean
  temppred <- temppred * tempdepmean
  #Remove temporary objects from memory before next cycle of loop
  rm(tempmeanloc, tempdepmean)
 }

 #Store the temporary results frame in a temporary output frame
 tempout <- tempdata
 #Replace the observed and predicted values of the dependent variable in the temporary output frame with their unlogged (and perhaps un mean scaled) values
 tempout[, 3] <- tempobs
 tempout[, 4] <- temppred
 #Rename the columns of the temporary output frame appropriately
 names(tempout) <- c(idvar, tvar, "observed", "predicted")
 #Store the temporary output in a final output component for inclusion in the list of results for output
```

```
tempbench <- tempout
#Calculate the percent difference between the observed and predicted values (logarithmically)
tempbench$diff <- log(tempbench$observed / tempbench$predicted)

#Round observed, predicted, and difference values appropriately
tempbench$observed <- round(tempbench$observed, 0)
tempbench$predicted <- round(tempbench$predicted, 0)
tempbench$diff <- round(tempbench$diff, rdigits)

#Check if tstat is true, and if so, add columns to the final output component containing the t statistics and p values for the differences
if (tstat == T) {
 tempbench[, 6] <- temptvals
 tempbench[, 7] <- temppvals
 names(tempbench)[6:7] <- c("tstat", "pvalue")
 #Round the columns appropriately and fix 0 pvalue results
 tempbench$tstat <- round(tempbench$tstat, rdigits)
 tempbench$pvalue <- round(tempbench$pvalue, rdigits)
 p0locs <- which(tempbench$pvalue == 0)
 tempbench$pvalue <- as.character(tempbench$pvalue)
 pvalchars <- nchar(tempbench$pvalue)
 pval0s <- rdigits + 2 - pvalchars

 for (j in 1:length(pval0s)) {
  tempbench$pvalue[j] <-
   ifelse(
    nchar(tempbench$pvalue[j]) != rdigits + 2,
    paste(
     tempbench$pvalue[j],
```

```
          paste(rep(0, pval0s[j]), sep = "", collapse = ""),
          sep = ""
        ),
        tempbench$pvalue[j]
      )
    }

    if (length(p0locs) > 0) {
      r0val <- paste("<", 10 ^ -rdigits, sep = "")
      tempbench$pvalue[p0locs] <- r0val
      rm(r0val)
    }
    rm(p0locs, pvalchars, pval0s)
  }
  #Remove temporary objects from memory before next cycle of loop
  rm(tempobs, temppred, temptvals, temppvals, tempdata)

  #Check whether company name was specified, and, if so, add a column to the final output component containing the name of the company being benchmarked
  if (length(compname) > 0) {
    tempbench$company <- compname
    #Rearrange the columns so that company is the second column is the benchmarking results data frame
    tempbenchnewcols <- c(1, dim(tempbench)[2], 2:(dim(tempbench)[2] - 1))
    tempbench <- tempbench[, tempbenchnewcols]
    #Remove temporary objects from memory before next cycle of loop
    rm(tempbenchnewcols)
  }
```

```
    #Store the benchmarking results data frame for the current model in the list of result sets for output, and name it appropriately
    outbenchmarks[[i]] <- tempbench
    #Name the current benchmarking results appropriately, using the name of the benchmarked company if it is provided
    names(outbenchmarks)[i] <-
      paste(names(models)[i], " - ", "Performance of ", companyid, sep = "")
    if (length(compname) > 0) {
      names(outbenchmarks)[i] <-
        paste(names(models)[i], " - ", "Performance of ", compname, sep = "")
    }


    #Remove temporary objects from memory before next cycle of loop
    rm(
      tempresults,
      tempmodel,
      tempactype,
      temprhometh,
      temphetmeth,
      tempdepmark,
      tempdepvarname,
      tempindvarnamesm,
      tempbench,
      tempout,
      tempbenchdata,
      tempbenchlocs,
      tempformula,
      tempoosdata,
      tempooslocs,
```

```
      tempoosmodel
    )

    #Reset row names of current benchmarking results to their default
    row.names(outbenchmarks[[i]]) <-
      as.character(1:length(row.names(outbenchmarks[[i]])))

  }
  #Remove temporary objects from memory
  rm(idvarmark, tvarmark)
  #Return list of benchmarking results
  return(outbenchmarks)
}
```

```
#the following function calculates elasticities and marginal costs for a set of user-specified outputs
#first run, from windows command line:
#1. cd C:\Users\Alex\Documents\R\win-library\3.3\Ryacas\yacdir
#2. yacas --server 9734
#the arguments are as follows:
# 1) models is the list of models based on which marginal costs are to be produced
# 2) alldata is the full data set used to regress the models
# 3) origdata is the original data set (not transformed in any way, ie mean-scaled or logged) that was used to produce alldata
# 4) idvar is the name of the variable that uniquely identifies companies in the sample
# 5) tvar is the name of the variable that uniquely identifies time periods in the sample
```

6) alloutputs is a character vector containing the full set of outputs used across models

(all outputs do not have to be included in all models; models can have different sets of outputs included;

however, no variable can appear as an output in one model but a business condition in another)

7) firmave is a binary variable indicating whether average elasticities and marginal costs should be calculated.

the default is false.

if false, elasticities and marginal costs for every company-year observation in the sample are returned. if true,

average elasticities and marginal costs for each company are returned.

#all the following arguments are only meaningful if firmave is true

8) varinfo is the informational data frame returned by logvars during the production of alldata

9) varmeans is the informational data frame returned by meanscale during the production of alldata

10) metype is the type of average used to calculate the elasticities.

the options for metype are "ame" (Average Marginal Effects) and "mem" (Marginal Effects at Means).

this affects the way in which the elasticities are calculated.

when metype is "ame", elasticites are calculated at every company year observation and then averaged for each company.

when metype is "mem" company-specific means are calculated for all relevant variables, and then elasticites are

calculated at those company-specific means. the default is "ame".

11) amemeanorder affects how marginal effects are transformed into marginal costs when metype is specified as "ame".

this affects the way in which marginal costs are calculated.

the options for amemeanorder are "mean" and "marg". when amemeanorder is "mean", marginal costs are calculated as

average elasticites transformed at firm mean values.

when it is "marg", marginal costs are calculated as the average of the elasticities transformed

at the firm-year values. the default is "mean".

12) meantype is the way in which the means of the variables in alldata are calculated for each firm

```
#     when metype is specified as "mem". this affects the way in which the elasticities are calculated.

#     the options for meantype are "meanlog" and "logmean". when meantype is "meanlog", firm specific means

#     for all relevant variables are calculated based on the transformed dataset, alldata.

#     when meantype is "logmean", firm specific means for all relevant variables are calculated

#     based on the original dataset, origdata, and then mean-scaled and logged according to the information in varmeans and varinfo. the default is "meanlog".

# 13) sereport indicates whether delta method standard errors should be included in the output data

#     frame for the elasticities. the default is true.



bmelastmarg <- function(models,
                        alldata,
                        origdata,
                        idvar,
                        tvar,
                        alloutputs,
                        firmave = F,
                        varinfo = NULL,
                        varmeans = NULL,
                        metype = "ame",
                        meantype = "meanlog",
                        amemeanorder = "mean",
                        sereport = T,
                        namevar = NULL,
                        verbose = F) {
  #mark the position of the id variable
  idvarmark <- which(names(alldata) == idvar)
  #mark the position of the id variable
```

```
  tvarmark <- which(names(alldata) == tvar)

  #initialize lists of results
  outputelastlist <- list(NULL)
  outputmarglist <- list(NULL)

  #loop over models
  for (i in 1:length(models)) {
    #store the current model and both datasets in temporary objects
    tempmodel <- models[[i]]
    tempdata <- alldata
    temporig <- origdata

    #print the current model
    if (verbose == T) {
      print(tempmodel)
    }

    #produce a dataframe of company means from the original, untransformed data
    #first get the column containing company ids
    idlisto <- temporig[, idvarmark]
    #now create a new, deidentified data frame of the original data
    unidorigdata <- temporig[, -c(idvarmark, tvarmark)]
    #check which columns in the deidentified data are numeric and reduce the deidentified data set to
only those
    numtesto <- sapply(unidorigdata, is.numeric)
    nummarko <- which(numtesto == T)
    unidorigdata <- unidorigdata[, nummarko]
```

```
    #produce a new data frame of company means
    origmeanframe <- aggregate(unidorigdata, by = list(idlisto), FUN = mean)
    #rename the grouping variable appropriately and reorder the data
    names(origmeanframe)[1] <- idvar
    origmeanframe <- origmeanframe[order(origmeanframe[, 1]), ]



    #produce a dataframe of company means from the transformed data used in the regression
    #first get the column containing company ids
    idlistt <- tempdata[, idvarmark]
    #now create a new, deidentified data frame of the transformed data
    unidtempdata <- tempdata[, -c(idvarmark, tvarmark)]
    #check which columns in the deidentified data are numeric and reduce the deidentified data set to
only those
    numtestt <- sapply(unidtempdata, is.numeric)
    nummarkt <- which(numtestt == T)
    unidtempdata <- unidtempdata[, nummarkt]


    #produce a new data frame of company means
    tempmeanframe <- aggregate(unidtempdata, by = list(idlistt), FUN = mean)
    #rename the grouping variable appropriately and reorder the data
    names(tempmeanframe)[1] <- idvar
    tempmeanframe <- tempmeanframe[order(tempmeanframe[, 1]), ]



    #Get the dependent variable name for the model currently in use
    tempdepvarname <- unlist(strsplit(names(models)[i], " - "))[1]
```

```
  #Get the names of the right hand side variables currently in use for the model
  tempindvarnames <- names(tempmodel$coefficients[-1])

  tempbetas <- tempmodel$coefficients
  tempvcov <- tempmodel$vcov

  #This section of the code is included for handling interactive and squared terms included in the model but not found in the data frame
  #This section identifies such terms, renames them systematically, and creates columns for them in the tempoosdata and tempbenchdata data frames
  #These steps are necessary to ensure proper handling of matrix operations later in the code for calculating t statistics

  #Loop over the names of all independent variables
  for (k in 1:length(tempindvarnames)) {
   #Intialize a temporary marker variable for whether the independent variable under consideration is a squared or interaction term
    tempcheck <- 0
    #Check whether the independent variable under consideration is an squared term
    tempIcheck <- ifelse(substr(tempindvarnames[k], 1, 2) == "I(", 1, 0)

    #If the independent variable under consideration is a squared term, standardize its naming to match the naming of interactive terms in R's coefficient listings
    if (tempIcheck == 1) {
     tempIterm <- tempindvarnames[k]
     #Cut the "I(" and ")" off either end of the squared term's string components
     tempIterm <- substr(tempIterm, 3, nchar(tempIterm) - 1)
     #Replace the old name of the squared term with a new variable name whose form matches the names of interactive terms
     tempindvarnames[k] <- tempIterm
```

```
    #Remove the components of the squared term from memory
    rm(tempIterm)
  }

  #Check whether the independent variable currently under consideration is a squared term or an interactive term
  tempcheck <-
    ifelse(length(unlist(strsplit(
      tempindvarnames[k], ":"
    ))) > 1, 1, 0)

  #If the independent variable currently under consideration is a squared term or an interactive term, rename it so that R can parse it as an expression (replace ":" with "*")
  if (tempcheck == 1) {
    tempIterm <- tempindvarnames[k]
    tempIterm <- gsub(":", "*", tempIterm)
    tempindvarnames[k] <- tempIterm
  }
}

allmodelvars <- unique(unlist(strsplit(tempindvarnames, "*", fixed = T)))
incoutputmarks <- which(alloutputs %in% allmodelvars)
outputs <- alloutputs[incoutputmarks]

#rename coefficients to ensure proper handling by deltamethod
names(tempbetas) <-
  c("x1", paste("x", 2:(length(
    tempindvarnames
  ) + 1), sep = ""))
```

```
  #add intercept term to list of independent variables
  tempindvarnames <- c("1", tempindvarnames)

  #rename variance covariance matrix rows and columns appropriately
  colnames(tempvcov) <- names(tempbetas)
  rownames(tempvcov) <- names(tempbetas)

  #produce the list of terms to be added together in the model
  tempindvarterms <- paste(names(tempbetas), tempindvarnames, sep = "*")
  #initialize the expression for the right hand side of the model specification
  temprhsexp <- tempindvarterms[1]

  #If there is more than one right hand side variable, complete the expression for the current model
  for (j in 2:length(tempindvarterms)) {
    temprhsexp <- paste(temprhsexp, tempindvarterms[j], sep = "+")
  }

  #initialize lists of partial derivatives for use in calculating elasticities and their standard errors
  tempmederivlist <- list()
  tempmesederivlist <- list()

  #loop over outputs
  for (j in 1:length(outputs)) {
    #get the output currently under consideration
    tempop <- outputs[j]
    #produce the yacas expression for the derivative of the current model specification with respect to
the current output
    tempderiv1exp <- paste("D(", tempop, ")", temprhsexp, sep = "")
```

```
    #print the yacas expression for the derivative of the current model specification with respect to the current output
    if (verbose == T) {
      print(tempderiv1exp)
    }
    #pass the derivative to yacas and get the character class result
    tempderiv1 <- yacas(tempderiv1exp)
    tempderiv1 <- as.character(tempderiv1)
    #store the derivative in the temporary objects for both lists of partial derivatives
    tempmederiv <- tempderiv1
    tempmesederiv <- tempderiv1
    #loop over the number of terms in the model (looping is done from highest to lowest to ensure that "x10" is not recognized as "x1" by gsub)
    for (k in length(tempindvarterms):1) {
      #replace the beta placeholders (currently x1, x2, etc with their true values)
      tempmederiv <- gsub(names(tempbetas)[k], tempbetas[k], tempmederiv)
    }
    #store the modified temporary objects for the partial derivatives in both lists
    tempmederivlist[[j]] <- tempmederiv
    tempmesederivlist[[j]] <- tempmesederiv
  }


  #initialize the list of variance covariance matrices
  tempmevcovlist <- list(NULL)


  #everything above this point in the program will be performed no matter what type of elasticities and marginal costs are requested
  #at this point, the program begins to check those options and perform different tasks based on what the user has requested
```

```
  #the following section will be performed whenever company-year specific values are requested, or when the marginal effects asked for are Average Marginal Effects
  if (metype == "ame" | firmave == F) {
    #create the data frame for the yearly elasticities
    tempeloutputy <- tempdata[, c(idvarmark, tvarmark)]

    #if marginal costs is to be calculated as the average marginal cost of each firm, or yearly, create the data frame for the yearly marginal costs
    if (amemeanorder == "marg" | firmave == F) {
      tempmargoutputy <- tempdata[, c(idvarmark, tvarmark)]
    }
    #if marginal cost is to be calculated as the marginal cost at the average elasticity for each firm, create the data frame for the firm marginal costs
    if (amemeanorder == "mean" & firmave == T) {
      tempmargoutput <- tempmeanframe[, 1]
      tempmargoutput <- as.data.frame(tempmargoutput)
    }

    #loop over outputs
    for (j in 1:length(outputs)) {
      #get the output currently under consideration
      tempop <- outputs[j]
      #get the derivative that defines the current output's elasticity
      tempmederiv <- tempmederivlist[[j]]
      #get the derivative to be used to calculate delta method standard errors for the elasticities
      tempsederiv <- tempmesederivlist[[j]]
      #create a copy of the SE derivative for each company-year observation
      tempsederivs <- as.list(rep(tempsederiv, dim(tempdata)[1]))
      #loop over company-year observations
      for (k in 1:dim(tempdata)[1]) {
```

```
#get the row currently under consideration
temprow <- tempdata[k, ]
#get the SE derivative to be modified
tempsederivrow <- tempsederivs[[k]]
#turn the current SE derivative into a one-sided formula to ensure proper handling by deltamethod
tempsederivrow <- paste("~", tempsederivrow)
#loop over the number of terms in the model (looping is done from highest to lowest to ensure that "x10" is not recognized as "x1" by gsub)
for (l in length(tempindvarnames):1) {
 #when the intercept is being considered, just replace it with "x1"
 if (l == 1) {
  tempname <- tempindvarnames[l]
  tempbetaname <- names(tempbetas)[l]
  tempoldval <- paste(tempbetaname, tempname, sep = "*")
  tempnewval <- "x1"
  tempsederivrow <- gsub(tempoldval, tempnewval, tempsederivrow)
 }
 #for terms other than the intercept, replace the instances of each independent variable found in the current formula with the value of that variable according to the data in the current row
 if (l != 1) {
  #get name of independent variable currently under consideration
  tempname <- tempindvarnames[l]
  #get name of coefficient currently under consideration
  tempbetaname <- names(tempbetas)[l]
  #store the name of the current independent variable as the old value to be replaced
  tempoldval <- tempname
  #calculate the value of the current independent variable according to the row under consideration
  tempnewval <- eval(parse(text = tempname), temprow)
```

```
        #replace the independent variable under consideration with its numeric value in the formula currently under consideration
          tempsederivrow <- gsub(tempoldval, tempnewval, tempsederivrow)
        }
        #remove temporary objects from memory before next cycle of loop
        rm(tempname, tempbetaname, tempoldval, tempnewval)
      }
      #replace the formula currently under consideration in the list of SE derivatives with its modified counterpart
      tempsederivs[[k]] <- tempsederivrow
      #remove temporary objects from memory before next cycle of loop
      rm(temprow, tempsederivrow)
    }
    #convert character class SE derivatives to formulas
    tempsederivs <- lapply(tempsederivs, as.formula)
    #print the formula used to calculated elasticities for the current output
    if (verbose == T) {
      print(tempmederiv)
    }
    #calculate elasticities for each company-year observation in the data set
    tempelast <- eval(parse(text = tempmederiv), tempdata)
    #calculate standard errors for the elasticities
    tempelsey <- deltamethod(tempsederivs, tempbetas, tempvcov, ses = T)
    #get the variance covariance matrix for the elasticities
    tempelvcov <- deltamethod(tempsederivs, tempbetas, tempvcov, ses = F)

    #store the elasticities in a new, temporary column in the company-year elasticity frame
    tempeloutputy$elastnew <- tempelast
    #rename the temporary column appropriately
```

```
    names(tempeloutputy)[dim(tempeloutputy)[2]] <-
      paste("el", tempop, sep = "_")
    #if company-year elasticites are to be reported, store the delta method standard errors in a new, temporary column in the elasticity output frame and then rename that column appropriately
    if (firmave == F) {
      tempeloutputy$elastnewse <- tempelsey
      names(tempeloutputy)[dim(tempeloutputy)[2]] <-
        paste("el", tempop, "se", sep = "_")
    }
    #if firm average elasticites are to be reported, store the elasticity delta method variance covariance matrix in the list of elasticity variance covariance matrices
    if (firmave == T) {
      tempmevcovlist[[j]] <- tempelvcov
    }
    #remove temporary objects from memory before next cycle of loop
    rm(tempelast, tempmederiv)
  }

  #if company-year elasticities and marginal costs are to be reported, or if average marginal costs are to be reported, calculate marginal costs for all company-year observations
  if (firmave == F | amemeanorder == "marg") {
    #loop over outputs
    for (j in outputs) {
      #mark the output currently under consideration in the original, untransformed data
      outputmark <- which(names(origdata) == j)
      #mark the elasticity of that output
      tempelastmark <- which(names(tempeloutputy) == paste("el", j, sep = "_"))
      #mark the dependent variable in the original, untransformed data
      tempdepmark <- which(names(origdata) == tempdepvarname)
      #calculate the marginal cost for each each company-year observation
```

```
      tempmarg <-
        tempeloutputy[, tempelastmark] * origdata[, tempdepmark] / origdata[, outputmark]
      #store the marginal costs in a new, temporary column
      tempmargoutputy$margnew <- tempmarg
      #rename the temporary column appropriately
      names(tempmargoutputy)[dim(tempmargoutputy)[2]] <-
        paste("margcost", j, sep = "_")
    }


  }
  #if company-year elasticities are to be reported, store the company-year elasticites and marginal costs in the true output frames
  if (firmave == F) {
    tempeloutput <- tempeloutputy
    tempmargoutput <- tempmargoutputy
  }
  #if firm average elasticites and marginal costs are to be reported, create the appropriate vectors and data frames
  if (firmave == T) {
    #store the id variable from the elasticity output frame
    tempidliste <- tempeloutputy[, 1]
    #store vector of unique values of the id variable in the elasticity output frame
    tempidlistse <- unique(tempidliste)
    #turn both vectors into data frames and rename their columns appropriately
    tempeloutput <- data.frame(varid = tempidlistse)
    tempelseoutput <- data.frame(varid = tempidlistse)
    names(tempeloutput)[1] <- idvar
    names(tempelseoutput)[1] <- idvar
```

```
#loop over outputs
for (j in 1:length(outputs)) {
 #get the output currently under consideration
 tempop <- outputs[j]
 #intialize temporary columns in the elasticity and elasticity standard error output frames
 tempeloutput$newelast <- 0
 tempelseoutput$newelastse <- 0
 #mark the elasticity currently under consideration
 tempelastmark <-
  which(names(tempeloutputy) == paste("el", tempop, sep = "_"))
 #get the delta method variance covariance matrix for the elasticites currently under consideration
 tempmevcov <- tempmevcovlist[[j]]
 #loop over companies
 for (k in 1:length(tempidlistse)) {
  #get the copmany currently under consideration
  tempid <- tempidlistse[k]
  #get the rows in the company-year id listing that correspond to the the company currently under
consideration
  templocs <- which(tempidliste == tempid)
  #get the yearly elasticity values for the company currently under consideration
  tempelvals <- tempeloutputy[templocs, tempelastmark]
  #reduce the variance covariance matrix to the entries that correspond to the current company's
yearly elasticities
  tempelvcov <- tempmevcov[templocs, templocs]
  #create dummies for use in the formula to calculate firm average elasticities
  tempeldums <- paste("x", 1:length(templocs), sep = "")
  #intialize formula for calculation of the current firm's average elasticity
  tempelform <- paste("~1/", length(templocs), "*(", tempeldums[1], sep =
                      "")
```

```
        #fill the formula for calculation of the current firm's average elasticity
        for (l in 2:length(templocs)) {
          tempelform <- paste(tempelform, tempeldums[l], sep = "+")
        }
        #complete the formula for calculation of the current firm's average elasticity
        tempelform <- paste(tempelform, ")", sep = "")
        #convert the character class formula to formula class
        tempelform <- as.formula(tempelform)
        #calculate the delta method standard error for the average elasticity of the firm currently under consideration
        tempelse <- deltamethod(tempelform, tempelvals, tempelvcov, ses = T)
        #calculate the average elasticity of the firm currently under consideration
        tempelval <- mean(tempelvals)
        #store the firm average elasticity and its standard error in the appropriate columns in the output data frames
        tempeloutput$newelast[k] <- tempelval
        tempelseoutput$newelastse[k] <- tempelse
      }
      #rename the output data frame columns approriately
      names(tempeloutput)[dim(tempeloutput)[2]] <-
        paste("el", tempop, sep = "_")
      names(tempelseoutput)[dim(tempelseoutput)[2]] <-
        paste("el", tempop, "se", sep = "_")

    }
    #reorder the output data frame rows to ensure they match
    tempeloutput <- tempeloutput[order(tempeloutput[, 1]), ]
    tempelseoutput <- tempelseoutput[order(tempelseoutput[, 1]), ]
    #bind the elasticity and elasticity standard error frames together
```

```
tempeloutput <- cbind(tempeloutput, tempelseoutput[, -1])


#reorder the columns in the elasticity output data frame so that elasticity standard errors appear directly to the right of the values to which they correspond
newcols <- sort(c(1, rep(2:(
  length(outputs) + 1
), 2)))
newcols[seq(3, length(newcols), 2)] <-
  newcols[seq(3, length(newcols), 2)] + length(outputs)
tempeloutput <- tempeloutput[, newcols]


#if firm average marginal costs are to be calculated as the average of yearly marginal costs, average the company-year marginal cost data frame by company
if (amemeanorder == "marg") {
  #create vector containing the id variable column from the marginal cost data frame
  tempidlistm <- tempmargoutputy[, 1]
  #remove the company and year columns from the marginal cost data frame
  tempmargoutputy <- tempmargoutputy[, -c(1:2)]
  #average the company-year marginal cost data frame by company
  tempmargoutput <-
    aggregate(tempmargoutputy,
              by = list(tempidlistm),
              FUN = mean)
  #rename the grouping variable appropriately
  names(tempmargoutput)[1] <- idvar
}
#if firm average marginal costs are to be calculated as the marginal cost at the average elasticity, use the firm average elasticities and the data frame of variable means from the original data set to calculate the appropriate values
if (amemeanorder == "mean") {
```

```
        #loop over outputs
        for (j in outputs) {
          #mark the position of the output currently under consideration in the data frame of means from the original data
          outputmark <- which(names(origmeanframe) == j)
          #mark the position of the elasticity currently under consideration
          tempelastmark <- which(names(tempeloutput) == paste("el", j, sep = "_"))
          #mark the position of the dependent variable in the data frame of means from the original data
          tempdepmark <- which(names(origmeanframe) == tempdepvarname)
          #calculate the marginal cost for each company
          tempmarg <-
            tempeloutput[, tempelastmark] * origmeanframe[, tempdepmark] / origmeanframe[, outputmark]
          #store the marginal costs in a new, temporary column in the marginal cost output data frame
          tempmargoutput$margnew <- tempmarg
          #rename the temporary column appropriately
          names(tempmargoutput)[dim(tempmargoutput)[2]] <-
            paste("margcost", j, sep = "_")
        }
        #rename the id variable column appropriately
        names(tempmargoutput)[1] <- idvar
      }

    }

  }

  #the following section of this code will only ever be performed when firm average marginal costs are requested and the type of elasticity to be calculated is specified as "Marginal Effects at Means"
  if (metype == "mem" & firmave == T) {
```

```
    #if the data to be used in calculating elasticities and marginal costs are the means of the original data, get that data and transform it exactly as the data used in the original regressions was transformed
    if (meantype == "logmean") {
      #replace the data frame of the means of the transformed data with the data frame of means of the original data
      tempmeanframe <- origmeanframe
      #mean scale the original data using the information provided in the varmeans argument
      #loop over the rows of varmeans
      for (j in 1:dim(varmeans)[1]) {
        #store the name of the variable currently under consideration in a temporary object
        tempvar <- varmeans$varname[j]
        #store the mean of the variable currently under consideration in a temporary object
        tempmean <- varmeans$varmean[j]
        #mark the position of the variable currently under consideration in the new regression data frame
        tempvarmark <- which(names(tempmeanframe) == tempvar)
        #if that variable exists in the regression data frame, mean scale it using the mean provided in varmeans
        if (length(tempvarmark) > 0) {
          tempmeanframe[, tempvarmark] <- tempmeanframe[, tempvarmark] / tempmean
        }
        #remove temporary objects from memory before next cycle of loop
        rm(tempvar, tempmean, tempvarmark)
      }
    #log-transform the mean scaled original data using the information provided in the varinfo argument
      #loop over the rows of varinfo
      for (j in 1:dim(varinfo)[1]) {
        #store the name of the variable currently under consideration in a temporary object
        tempvar <- varinfo$varname[j]
        #store the original check for the presence of zero values in a temporary object
```

```
      tempcheck0 <- varinfo$check0[j]
      #mark the position of the variable currently under consideration in the new regression data frame
      tempvarmark <- which(names(tempmeanframe) == tempvar)
      #if that variable exists in the regression data frame, log-transform it using the information provided in varinfo
      if (length(tempvarmark) > 0) {
       #if there were no zero values in the original variable, log the variable
       if (tempcheck0 == 0) {
        tempmeanframe[, tempvarmark] <- log(tempmeanframe[, tempvarmark])
       }
       #if there were zero values in the original variable, add one to it and the log it
       if (tempcheck0 == 1) {
        tempmeanframe[, tempvarmark] <- log(tempmeanframe[, tempvarmark] + 1)
       }
      }
      #remove temporary objects from memory before next cycle of loop
      rm(tempvar, tempcheck0, tempvarmark)
     }
    }

    #create the output data frames
    tempeloutput <- tempmeanframe[, 1]
    tempeloutput <- as.data.frame(tempeloutput)
    tempelseoutput <- tempmeanframe[, 1]
    tempelseoutput <- as.data.frame(tempelseoutput)
    tempmargoutput <- tempmeanframe[, 1]
    tempmargoutput <- as.data.frame(tempmargoutput)

    #loop over outputs
```

```
  for (j in 1:length(outputs)) {
    #get the output currently under consideration
    tempop <- outputs[j]
    #get the formula for the elasticity of that output
    tempmederiv <- tempmederivlist[[j]]
    #get the formula to be used in calculating delta method standard errors for the elasticites
    tempsederiv <- tempmesederivlist[[j]]
    #replicate the SE formula for each row in the new regression data frame
    tempsederivs <- as.list(rep(tempsederiv, dim(tempmeanframe)[1]))
    #loop over rows in the new regression data frame (i.e. over companies)
    for (k in 1:dim(tempmeanframe)[1]) {
      #get the row of data currently under consideration
      temprow <- tempmeanframe[k, ]
      #get the corresponding SE derivative formula
      tempsederivrow <- tempsederivs[[k]]
      #convert the current SE derivate formula to a one-sided formula
      tempsederivrow <- paste("~", tempsederivrow)
      #loop over the number of terms in the model (looping is done from highest to lowest to ensure
that "x10" is not recognized as "x1" by gsub)
      for (l in length(tempindvarnames):1) {
        #replace the intercept term with "x1"
        if (l == 1) {
          tempname <- tempindvarnames[l]
          tempbetaname <- names(tempbetas)[l]
          tempoldval <- paste(tempbetaname, tempname, sep = "*")
          tempnewval <- "x1"
          tempsederivrow <- gsub(tempoldval, tempnewval, tempsederivrow)
        }
```

```
        #replace instances of all other independent variables in the current SE derivative formula with the values according to the row of data currently under consideration
        if (l != 1) {
          #get the name of the independent variable currently under consideration
          tempname <- tempindvarnames[l]
          #get the name of the coefficient currently under consideration
          tempbetaname <- names(tempbetas)[l]
          #store the name of the independent variable currently under consideration as the value to be replaced
          tempoldval <- tempname
          #get the value of the independent variable currently under consideration according to the corresponding row
          tempnewval <- eval(parse(text = tempname), temprow)
          #replace all instances of the current independent variable's name with its numeric value
          tempsederivrow <- gsub(tempoldval, tempnewval, tempsederivrow)
        }
        #remove temporary objects from memory before next cycle of loop
        rm(tempname, tempbetaname, tempoldval, tempnewval)
      }
      #replace the item currently under consideration in the listing of SE derivatives with its modified counterpart
      tempsederivs[[k]] <- tempsederivrow
      #remove temporary objects from memory before next cycle of loop
      rm(temprow, tempsederivrow)
    }
    #convert character class formulas to formula class to ensure proper handling by delta method
    tempsederivs <- lapply(tempsederivs, as.formula)
    #calculate the elasticities for each company
    tempelast <- eval(parse(text = tempmederiv), tempmeanframe)
    #calculate the standard errors for those elasticities
```

```
    tempelse <- deltamethod(tempsederivs, tempbetas, tempvcov, ses = T)


    #store the elasticities for the output currently under consideration in a new, temporary column
    tempeloutput$elastnew <- tempelast
    #rename the temporary column appropriately
    names(tempeloutput)[dim(tempeloutput)[2]] <-
      paste("el", tempop, sep = "_")
    #store the standard errors of elasticities for the output currently under consideration in a new,
temporary column
    tempeloutput$elastnewse <- tempelse
    #rename the temporary column appropriately
    names(tempeloutput)[dim(tempeloutput)[2]] <-
      paste("el", tempop, "se", sep = "_")


    #remove temporary objects from memory before next cycle of loop
    rm(tempelast, tempmederiv)


  }
  #loop over outputs
  for (j in outputs) {
    #mark the position of output currently under consideration in the data frame containing the means
of the original data
    outputmark <- which(names(origmeanframe) == j)
    #mark the position of the elasticity currently under consideration
    tempelastmark <- which(names(tempeloutput) == paste("el", j, sep = "_"))
    #mark the position of the dependent variable in the data frame containing the means of the original
data
    tempdepmark <- which(names(origmeanframe) == tempdepvarname)
    #calculate the marginal costs for the output currently under consideration
    tempmarg <-
```

```
      tempeloutput[, tempelastmark] * origmeanframe[, tempdepmark] / origmeanframe[, outputmark]
    #store the marginal costs in a new, temporary column in the marginal costs output data frame
    tempmargoutput$margnew <- tempmarg
    #rename the temporary column appropriately
    names(tempmargoutput)[dim(tempmargoutput)[2]] <-
      paste("margcost", j, sep = "_")
  }
  #rename the id variable columns of both output data frames appropriately
  names(tempeloutput)[1] <- idvar
  names(tempmargoutput)[1] <- idvar

}

#get the sum of the elasticities and bind it to the elasticity data frame for output
ellocs <- which(names(tempeloutput) %in% paste("el", outputs, sep = "_"))
redelframe <- tempeloutput[, ellocs]
elsums <- apply(redelframe, MARGIN = 1, sum)
tempeloutput$elsum <- elsums

#check if standard errors are to be reported and remove them if sereport is F
if (sereport == F) {
  selocs <-
    which(names(tempeloutput) %in% paste("el", outputs, "se", sep = "_"))
  tempeloutput <- tempeloutput[, -selocs]
}

#check if a column with company names was specified and, if so, add company names to the output
data frames
if (length(namevar) > 0) {
```

```
    namevarmark <- which(names(alldata) == namevar)
    elnamelocs <- match(tempeloutput[, 1], alldata[, idvarmark])
    mcnamelocs <- match(tempmargoutput[, 1], alldata[, idvarmark])
    tempeloutput$company <- alldata[elnamelocs, namevarmark]
    tempmargoutput$company <- alldata[mcnamelocs, namevarmark]
    tempeloutput <-
      tempeloutput[, c(1, dim(tempeloutput)[2], 2:(dim(tempeloutput)[2] - 1))]
    tempmargoutput <-
      tempmargoutput[, c(1, dim(tempmargoutput)[2], 2:(dim(tempmargoutput)[2] -
                                                         1))]
  }


  #reorder elasticity and and marginal cost output frames by the sum of elasticities
  neworder <-
    order(tempeloutput[, dim(tempeloutput)[2]], tempeloutput[, 1])
  tempeloutput <- tempeloutput[neworder, ]
  tempmargoutput <- tempmargoutput[neworder, ]

  #add the elasticity and marginal cost output data frames to their corresponding lists for output
  outputelastlist[[i]] <- tempeloutput
  outputmarglist[[i]] <- tempmargoutput

}



 #combine the lists containing elasticities and marginal costs into one list containing those two as
sublists
```

```
  alloutput <- list(outputelastlist, outputmarglist)
  #return the list of elasticities and marginal costs
  return(alloutput)
}
```